12:07:26 OCA PAD AMENDMENT - PROJECT HEADER INFORMATION 12/13/95

Active

Project #: C-36-X17 Cost share #: Rev #: 6
Center # : 10/24-6-R7917-0A0 Center shr #: OCA file #:
Work type : RES
Contract#: N00014-93-1-0783 Mod #: A00002 Document : GRANT
Prime #: Contract entity: GTRC

Subprojects ? : N CFDA: 12.300
Main project #: PE #: NA

Project unit: COMPUTING Unit code: 02.010.300
Project director(s):
SCHWAN K COMPUTING (404)894-2589

Sponsor/division names: NAVY / OFC OF NAVAL RESEARCH
Sponsor/division codes: 103 / 025

Award period: 930701 to 960615 (performance) 960615 (reports)

Sponsor amount	New this change	Total to date
Contract value	0.00	68,366.00
Funded	0.00	68,366.00
Cost sharing amount		0.00

Does subcontracting plan apply ?: N

Title: DPARTS - A DYNAMIC PARALLEL ADAPTIVE MULTIPROCESSOR REAL-TIME SCHEDULER

PROJECT ADMINISTRATION DATA

OCA contact: Jacquelyn L. Bendall 894-4820

Sponsor technical contact

ANDRE M. VAN TILBORG, CODE 3330
(703)696-4312

OFFICE OF NAVAL RESEARCH
BALLSTON TOWER ONE
800 NORTH QUINCY STREET
ARLINGTON, VA 22217-5660

Sponsor issuing office

SHARON MARSHALL, CODE 2520 (1520) SM
(703)696-2607

OFFICE OF CHIEF OF NAVAL RESEARCH
BALLSTON TOWER ONE
800 NORTH QUINCY STREET
ARLINGTON, VA 22217-5660

Security class (U,C,S,TS) : U ONR resident rep. is ACO (Y/N): Y
Defense priority rating : NA supplemental sheet
Equipment title vests with: Sponsor GIT X

Administrative comments -
ISSUED TO EXTEND THE PERIOD OF PERFORMANCE TO JUNE 15, 1996.

4

2 - SR562

GEORGIA INSTITUTE OF TECHNOLOGY
OFFICE OF CONTRACT ADMINISTRATION

NOTICE OF PROJECT CLOSEOUT

Closeout Notice Date 10/14/96

Project No. C-36-X17_____ Center No. 10/24-6-R7917-0A0_

Project Director SCHWAN K_____ School/Lab COMPUTING_____

Sponsor NAVY/OFC OF NAVAL RESEARCH_____

Contract/Grant No. N00014-93-1-0783_____ Contract Entity GTRC

Prime Contract No. _____

Title DPARTS - A DYNAMIC PARALLEL ADAPTIVE MULTIPROCESSOR REAL-TIME SCHEDULER__

Effective Completion Date 960615 (Performance) 960615 (Reports)

Closeout Actions Required:	Y/N	Date Submitted
Final Invoice or Copy of Final Invoice	Y	_____
Final Report of Inventions and/or Subcontracts	Y	_____
Government Property Inventory & Related Certificate	Y	_____
Classified Material Certificate	N	_____
Release and Assignment	N	_____
Other _____	N	_____

Comments_____
_____

Subproject Under Main Project No. _____

Continues Project No. _____

Distribution Required:

Project Director	Y
Administrative Network Representative	Y
GTRI Accounting/Grants and Contracts	Y
Procurement/Supply Services	Y
Research Property Managment	Y
Research Security Services	N
Reports Coordinator (OCA)	Y
GTRC	Y
Project File	Y
Other _____	N
_____	N

NOTE: Final Patent Questionnaire sent to PDPI.

0

C-36-X17
1

DPARTS – A Dynamic Parallel Adaptive Multiprocessor Real-Time Scheduler

PROGRESS REPORT 1

Byron Jeff
Karsten Schwan

College of Computing
Georgia Institute of Technology
Atlanta, GA 30332

1 Research Agenda

Motivation. The area of real-time scheduling has received increased attention during the last few years, in part due to the creation many recent multiprocessor and distributed applications demanding real-time performance. Such applications include robotics [5], multi-media [3] and music [2]. These applications are characterized by their need for real time response, heavy computational load, and unpredictable events that must be processed in a timely manner.

The specific application domain addressed by this research concerns multi-media systems. Specifically, it concerns the real-time synthesis of voice and music, where partially ordered sequences of computations and communications must be processed under given time constraints. We are investigating the dynamic (on-line) scheduling of action sequences triggered by real-time events, where multiple action sequences and different independent segments of a single action may execute in parallel. A real-time parallel digital audio synthesis system serves as a concrete example of such an application. A sample real-time event in this application is a note played on a Musical Instrument Device Interface (MIDI) keyboard.

The proposed DPARTS multiprocessor scheduler is now partially implemented. It will perform scheduling in an event driven manner, in reaction to sequences of incoming events. Furthermore, DPARTS will have the ability to adapt its scheduling during execution. Such adaptations will be performed in accordance with user specifications that address the actions to be taken in response to exceptions like deadline failures and sudden overloads. The intent is to have DPARTS adapt its schedule to maximize some global, time-dependent measure of schedule quality, even under high system loads.

Multi-media and Music Applications. Some recent research has addressed the scheduling and real-time processing of multi-media applications [3]. However, such research has addressed relatively static applications like the transfer of successive image frames across a network [1] or the scheduling of a music application with some fixed number of tasks [2]. In contrast, we are addressing multi-media applications that are highly dynamic, which means that applications must react to unexpected or unanticipated external events. Such reactions may result in the on-line creation of additional tasks that must be dynamically scheduled in reference to existing task sets. Furthermore, for dynamic music programs that react to unanticipated human inputs, timing constraints have to be stated differently from other application domains. Namely, it is not natural to state timing constraints in terms of start times, execution times, and deadlines. Instead, novel semantics have to be defined for 'deadlines', such as semantics that capture notions of lateness based on the perception of different musical instruments by the listener. For example, a deadline for an instrument like a drum approximates the well-known notion of hard deadlines [4] because drum beats must be very precise. On the other hand, for musical tones that are long and drawn out, such as those generated by string instruments, 'softer' deadline semantics are appropriate. We are investigating and specifying such novel timing semantics and using them in the development of efficient on-line scheduling algorithms and schedulers supporting this class of real-time applications.

Distributed Multiprocessor Scheduling. The scheduler being developed in this research is a specific example of an operating system service to be offered in future parallel and distributed systems. Part of this research will address the issues of distributed service design and implementation currently being addressed by other OS researchers. Specifically, we are building on the research described in [6] to construct the DPARTS scheduler to be internally concurrent so it can be easily scaled to different size parallel machines and to varying application demands. This implies that scheduling is performed by multiple concurrent and cooperating tasks.

DPARTS Scheduling Details. Some details about the DPARTS scheduler further demonstrate its novelty. As stated above, each external event may trigger a sequence of processes that jointly handle the event. These processes must execute in an application-specific order. DPARTS must have information about (1) such orderings, (2) the computation time of the involved processes, and (3) about possible alternate or optional processes. We are developing a directed graph representation to contain such information, where the computation times and any other information DPARTS needs to know about processes is stored in the graph's nodes, and the directed edges represent the processes' order of computation. Processes must be scheduled concurrently and to meet real-time constraints, such that the graph's topological order is maintained. This implies that multiple parts of the schedule graph may be active at any point during the applications execution.

The research questions addressed in this work focus on the development of fast heuristic algorithms, on experimentation with those algorithms, and on the use of such algorithms within the music application described above. We are not developing optimal algorithms since all of the scheduling and assignment problems we are addressing have been shown NP-hard. Some specific questions we are addressing include:

- When should DPARTS be invoked? What is the required frequency of DPARTS invocation with respect to the latency of scheduling decisions and the overheads incurred by scheduling?
- Should DPARTS have the option of rescheduling existing tasks if such rescheduling can result in the successful scheduling of otherwise unschedulable process sequences?
- Deadline semantics. In our sample music application, precise deadlines are usually not necessary in order to produce acceptable results. As such we are evaluating what type of sematics are appropriate for addressing the application we are scheduling.

2 DPARTS Status

The design of the scheduler is nearly completed, with implementation currently in progress. Through initial testing we have determined that the single largest degrader of performance is the Unix signal call. The elapsed time of catching and responding to a signal takes nearly half

of our proposed scheduling cycle. Based on these results our scheduler design is based on the non-preemptive execution of threads. The scheduler indicates to each application thread how many cycles it should execute before giving up the processor. Through the use of cooperative task scheduling we believe that much of the performace gained will be retained.

Preliminary responses to the above stated research questions are as follows:

- When should DPARTS be invoked? The DPARTS scheduler will be invoked once each execution frame. The execution frame time will be based on the Just Noticable Difference Perception Time for audio. We plan to start with a 5 millisecond cycle time. Also a DPARTS scheduler will be executed on each processor in a non-overlapping fashion throughout the system. Therefore, initially only one scheduler will be active in the system at any instant.

- What is the required frequency of DPARTS invocation with respect to the latency of scheduling decisions and the overheads incurred by scheduling? And, should DPARTS have the option of rescheduling existing tasks if such rescheduling can result in the successful scheduling of otherwise unschedulable process sequences? Initially, tasks will not be reschedulable. When all of the available time for a frame is filled, then the task invocation request will be passed to the next scheduler with an increasingly higher priority. In this manner, even low priority events such as string invocations will be scheduled in a reasonable time frame.

- Deadline semantics. In our sample music application, precise deadlines are usually not necessary in order to produce acceptable results. As such we will evaluate what type of sematics are appropriate for addressing the application we are scheduling. Furthermore, our initial model for incoming events will assign a relative priority to each different instrument. Within each time frame, available tasks will be scheduled in priority order and any remaining tasks will be passed to the next scheduler with a temporarily higher priority than the initial priority so that increasingly earlier events will be scheduled.

Digital Audio Synthesizer. Our sample application, a parallel digital audio synthesizer, consists of a number of interlinked modules that communicate by passing packets of audio samples. The application embodies modules to perform the following functions:

- Digital Audio Generation (sine wave, FM, wavetable lookup).

- Filtering (a generic digital filter that produces high pass, low pass, bandpass, and notch filter outputs).

- Amplification.

- Mixing.

- Effects (chorusing and delays).

- Output.

The user will specify the linkages between these modules, and the application will accept input events that triggers a set of modules which produce the synthesized digital audio output.

Digital Audio Synthesizer Status. The initial Digital Audio Generation, Mixing, and Output modules are complete. In addition, the Input Event Generator that converts MIDIFILES into a stream of incoming events has been completed.

Equipment Status. PCs are becoming increasingly important as actual business and research computing engines. For example, much of the research in Japan addressing real-time applications (the TRON project) is based on PCs. Therefore, our current equipment base is a PC purchased with equipment funds allocated to this work:

- 90 Mhz Pentium, 32 Megs RAM, 17 in SVGA Color Monitor.

This machine is attached via Ethernet to high performance computing and visualization engines, as well as to MIDI input devices and to multimedia output engines. The cost of the items are summarized in the attached expense report.

References

[1] D.P. Anderson and R.G. Herrtwich. Resource management for digital audio and video. In *Seventh IEEE Workshop on Real-Time Operating Systems and Software*, pages 99–103. IEEE, May 1990.

[2] D.P. Anderson and R. Kuivila. A system for computer music performance. *ACM Transactions on Computer Systems*, 8(1):56–82, Feb 1990.

[3] D.P. Anderson, S. Tzou, R. Wahbe, R. Govindan, , and M. Andrews. Support for continous media in the dash system. In *Proceedings of the 10th International Conference on Distributed Computing Systems, Paris France.* IEEE,ACM, 1990.

[4] Aloysius Ka-Lau Mok. *Fundamental Problems of Distributed Systems for the Hard Real-Time Environment.* PhD thesis, Laboratory for Computer Science, Massachussetts Institute of Technology, May 1983.

[5] Karsten Schwan and Rajiv Ramnath. Adaptable operating software for manufacturing systems and robots: A computer science research agenda. Technical report, Computer and Information Science, The Ohio State University, OSU-CISRC-TR-84-4, May 1984.

[6] Hongyi Zhou. *Task Scheduling and Synchronization for Multiprocessor Real-Time Systems.* PhD thesis, College of Computing, Georgia Institute of Technology, May 1992.

DPARTS – A Dynamic Parallel Adaptive Multiprocessor Real-Time Scheduler

FINAL REPORT

Byron Jeff
Karsten Schwan

College of Computing
Georgia Institute of Technology
Atlanta, GA 30332

Submitted to
Andre vanTilborg
Office of Naval Research (ONR)
800 N. Quincy St.
Arlington, VA 22217-5000.

Technical Contacts

Byron Jeff
Phone: (404) 880-6934
Email: byron@cc.gatech.edu

Karsten Schwan
Phone: (404) 894-2589
Email: schwan@cc.gatech.edu

1 Summary of Research Results

1.1 Motivation

The area of dynamic real-time scheduling has received increased attention during the last few years, in part due to the creation many recent multiprocessor and distributed applications demanding real-time performance. Such applications include robotics [5], multi-media [3], music [2], and complex command and control systems that execute complex mixtures of data dependent (e.g., target recognition) and human-initiated tasks. These applications are characterized by their need for real time response, potentially heavy computational load, and unpredictable events that must be processed in a timely manner.

The specific application domain addressed by this research concerns multi-media systems. Specifically, it concerns the real-time synthesis of voice and music, where partially ordered sequences of computations and communications must be processed under given time constraints. This grant has permitted us to investigate the dynamic (on-line) scheduling of action sequences triggered by real-time events, where multiple action sequences and different independent segments of a single action may execute in parallel. A real-time parallel digital audio synthesis system serves as a concrete example of such an application. A sample real-time event in this application is a note played on a Musical Instrument Device Interface (MIDI) keyboard.

The proposed DPARTS multiprocessor scheduler has been partially implemented and is performing scheduling in an event driven manner, in reaction to sequences of incoming events. The DPARTS system and the scheduler will have the ability to adapt scheduling during execution. Such adaptations will be performed in accordance with user specifications that address the actions to be taken in response to exceptions like deadline failures and sudden overloads. The intent is to have DPARTS adapt its schedule to maximize some global, time-dependent measure of schedule quality, even under high system loads. Toward this end, we are now also considering changes to the Unix operating system kernel to permit customizable scheduling supported by the kernel.

1.2 Multi-media and Music Applications

Some recent research has addressed the scheduling and real-time processing of multi-media applications [3]. However, such research has addressed relatively static applications like the transfer of successive image frames across a network [1] or the scheduling of a music application with some fixed number of tasks [2]. In contrast, we are addressing multi-media applications that are highly dynamic, which means that applications must react to unexpected or unanticipated external events. Such reactions will result in the on-line creation of additional tasks that must be dynamically scheduled in reference to existing task sets. Furthermore, for dynamic music programs that react to unanticipated human inputs, timing constraints have to be stated differently from other application domains. Namely, it is not reasonable to expect users to characterize timing constraints in terms of start times, execution times, and deadlines. Instead, we have defined some novel semantics for 'deadlines', such as semantics that capture notions of lateness based on the perception of different musical instruments by the listener. For example, a deadline for an instrument like a drum approximates the well-known notion of hard deadlines [4] because drum

beats must be very precise. On the other hand, for musical tones that are long and drawn out, such as those generated by string instruments, 'softer' deadline semantics are appropriate. We have begun to investigate and specify such novel timing semantics and are using them in the development of efficient on-line scheduling algorithms and schedulers supporting this class of real-time applications. What remains to be done is additional implementation so that we can compare our work to results attainable with 'classical' methods for on-line scheduling, such as deadline-based scheduling algorithms [7] or some of the extensions of rate-based scheduling addressing dynamic task arrivals [6].

Another aspect of the real-time generation of multimedia artifacts is the potential for a large reduction of the storage size of multimedia objects. Many of the elements of these objects can be represented by algorithms and event lists. An example of this is a MIDIfile representation of a piece of music. By storing/transporting the algorithms/event lists and generating the actual multimedia object in real time, a net reduction in the storage/transportation costs of such objects can be achieved.

1.3 Multiprocessor Scheduling

The scheduler developed in this research is a specific example of a customizable operating system service as offered in high performance systems, with more general methods for offering such services being developed in recent work, like the Exokernel research. Part of this research will address the issues designing such customizable services for parallel, not just sequential, machines, which is especially important due to the recent arrival of quad-processor Pentium boards. Such work is building on the research described in [7] and is constructing the DPARTS scheduler to be internally concurrent so it can be easily scaled to different size parallel machines and to varying application demands. This implies that scheduling is performed by multiple concurrent and cooperating tasks.

1.4 DPARTS Scheduling Details

Some details about the DPARTS scheduler further demonstrate its novelty. As stated above, each external event may trigger a sequence of processes that jointly handle the event. These processes must execute in an application-specific order. DPARTS must have information about (1) such orderings, (2) the computation time of the involved processes, and (3) about possible alternate or optional processes. We will develop a directed graph representation to contain such information, where the computation times and any other information DPARTS needs to know about processes is stored in the graph's nodes, and the directed edges represent the processes' order of computation. Processes must be scheduled concurrently and to meet real-time constraints, such that the graph's topological order is maintained. This implies that multiple parts of the schedule graph may be active at any point during the applications execution.

The specific research questions that remain to be addressed in this work focus on the development of fast heuristic algorithms, on experimentation with those algorithms, and on the use of such algorithms within the music application described above. We will not develop optimal algorithms since all of the scheduling and assignment problems we are addressing have been shown NP-hard. However, the

availability of the optimal multiprocessor real-time scheduling algorithm described in <hongyi> should enable us to compare our work with other rigorous research. Some specific questions we have begun to address are (see the preliminary responses to these questions listed in the next section):

1. When should DPARTS be invoked? What is the required frequency of DPARTS invocation with respect to the latency of scheduling decisions and the overheads incurred by scheduling?
2. Should DPARTS have the option of rescheduling existing tasks if such rescheduling can result in the successful scheduling of otherwise unschedulable process sequences?
3. Deadline semantics. In our sample music application, precise deadlines are usually not necessary in order to produce acceptable results. As such we will evaluate what type of sematics are appropriate for addressing the application we are scheduling.

1.5 DPARTS Scheduling Status

The design of the scheduler is completed. An initial prototype has been tested and with its evaluation, we have determined that the single largest degrader of performance is the Unix signal call. The elapsed time of catching and responding to a signal takes nearly half of our proposed scheduling cycle. Based on these results, our scheduler design is based on non-pre-emptive execution of threads. The scheduler indicates to each thread of the application how many execution cycles it should execute before giving up the processor. Through the use of cooperative task scheduling we believe that much of the performace gained will be retained. In addition, by using novel methods like 'processor reservation', we should be able to interact gracefully with the operating system kernel controlling the low-level events based on which scheduling decisions may have to be made.

Preliminary responses to the above stated research questions are as follows:

1. When should DPARTS be invoked? The DPARTS scheduler is invoked once each execution frame. The execution frame time is based on the Just Noticable Difference Perception Time for audio. We are starting with a 5 millisecond cycle time. Also a DPARTS scheduler is executed on each processor in a non-overlapping fashion throughout the system. So initially only on scheduler is active in the system at any instant.
2. What is the required frequency of DPARTS invocation with respect to the latency of scheduling decisions and the overheads incurred by scheduling?
3. Should DPARTS have the option of rescheduling existing tasks if such rescheduling can result in the successful scheduling of otherwise unschedulable process sequences? Initially tasks will not be reschedulable. When all of the available time for a frame is filled then the task invocation request will be passed to the next scheduler with an increasingly higher priority. In this manner even low priority events such as string invocations are scheduled in a reasonable time frame.

4. Deadline semantics. In our sample music application, precise deadlines are usually not necessary in order to produce acceptable results. As such we will evaluate what type of sematics are appropriate for addressing the application we are scheduling.

Our initial model assigns to incoming events a relative priority for each different instrument. Within each frame, available tasks are scheduled in priority order and any remaining tasks are passed to the next scheduler. An aging algorithm gives leftover tasks a temporarily higher priority than the initial priority, resulting in increasingly higher priorities for the scheduling of earlier events.

1.6 Digital Audio Synthesizer Details

Our sample application is a parallel digital audio synthesizer, which consists of a number of interlinked modules that communicate by passing packets containing audio samples. The application embodies modules that perform the following functions:

- Digital Audio Generation (sine wave, FM, wavetable lookup),
- Filtering (a generic digital filter that produces high pass, low pass, bandpass, and notch filter outputs),
- Amplification,
- Mixing,
- Effects (chorusing and delays), and
- Output.

The user specifies the linkages between these modules, and the application accepts input events that triggers a set of modules which produce the synthesized digital audio output.

1.7 Digital Audio Synthesizer Status

The initial Digital Audio Generation, Mixing, and Output modules are complete. In addition the Input Event Generator that converts MIDIFILES into a stream of incoming events has been completed.

2 Equipment Status

The initial proposed equipment needs failed to meet our goals due to the extremely heavy load on the existing SGI Multiprocessor. Instead, we chose to move to PC platforms, which are moving toward quad-processor boards (perhaps up to 16 in the near future). In response, the following items were purchased with this funding:

- 90 Mhz Pentium, 64Megs RAM, 17 in SVGA Color Monitor, 8X CDROM, multimedia speakers.

The cost of the items are summarized in the attached expense report.

In the immediate future, we intend to purchase a number of quad-processor Pentium machines, and to cluster them using high performance PCI-based interconnects (or even with distributed shared memory interconnects). This would result in an exciting set of equipment that would drive this research. In addition, we are investigating the use of Windows NT as a suitable research platform (we are now in the process of acquiring an NT source license).

3 Publications and Conferences/Workshops

DIMAS-Constructing Dynamic Interactive Multimedia Artifacts, B. Jeff and K. Schwan, Workshop Proceedings: THE ROLE OF REAL-TIME IN MULTIMEDIA/INTERACTIVE SYSTEMS, 1993

PARSYNTH: A Case Study on Implementing a Real-Time Digital Synthesize, B. Jeff and K. Schwan,Workshop Proceedings: 4th International Workshop on Parallel and Distributed Real-Time Systems, 1996

4 Interactions with Clark Atlanta University

As a faculty member at Clark Atlanta University, Byron Jeff assisted ONR related projects in several capacities:

Mentor Served as faculty mentor for ONR sponsored Summer Intership program at Georgia Tech. Duties included student selection and monitoring student research progress.

Advisor Served as faculty research advisor for Tiki Suarez, member of the ONR sponsored Prism-D program. Ms. Suarez graduated with honors and is now attending Florida State University as a PhD student in Computer Science.

5 Free software development

Another thrust of the project was the incorporation of the Free Software model into research. The Linux Operating System, a free Unixlike OS, has been heavily integrated into the research paradigm for this project. The usefulness is multifold:

Free As the software is freely available, students can duplicate research results without a heavy investment in resources.

Hardware Linux runs on standard PC hardware which is easily accessible to most researchers.

Source Since the source code for the OS kernel and all applications are available, it is possible to distribute modified versions of software along with applications.

References

[1] D.P. Anderson and R.G. Herrtwich. Resource management for digital audio and video. In *Seventh IEEE Workshop on Real-Time Operating Systems and Software*, pages 99–103. IEEE, May 1990.

[2] D.P. Anderson and R. Kuivila. A system for computer music performance. *ACM Transactions on Computer Systems*, 8(1):56–82, Feb 1990.

[3] D.P. Anderson, S. Tzou, R. Wahbe, R. Govindan, , and M. Andrews. Support for continous media in the dash system. In *Proceedings of the 10th International Conference on Distributed Computing Systems, Paris France.* IEEE,ACM, 1990.

[4] Aloysius Ka-Lau Mok. *Fundamental Problems of Distributed Systems for the Hard Real- Time Environment.* PhD thesis, Laboratory for Computer Science, Massachussetts Institute of Technology, May 1983.

[5] Karsten Schwan and Rajiv Ramnath. Adaptable operating software for manufacturing systems and robots: A computer science research agenda. Technical report, Computer and Information Science, The Ohio State University, OSU-CISRC-TR-84-4, May 1984.

[6] B. Sprunt, John P. Lehoczky, and Lui Sha. Exploiting unused periodic time for aperiodic service using the extended priority exchange algorithm. In *Proceedings of Real-Time Systems Symposium, Huntsville, AL*, pages 251–258. IEEE, 1988.

[7] Hongyi Zhou. *Task Scheduling and Synchronization for Multiprocessor Real-Time Systems.* PhD thesis, College of Computing, Georgia Institute of Technology, May 1992.

DIMAS - Constructing Dynamic Interactive Multimedia ArtifactS

Byron Jeff
Karsten Schwan

College of Computing
Georgia Institute of Technology
Atlanta, GA 30332

1 The DIMAS System

Real-time systems interact with their external execution environments, and complex, large-scale systems often cannot anticipate all such interactions. As a result, real-time systems are increasingly being constructed with mechanisms and policies that permit them to adjust to new environmental conditions, based on higher-level knowledge about system functionality and requirements[8, 4, 6, 5]. The goals of the DIMAS system are:

- to provide mechanisms, policies, and human interfaces for dynamic system configuration and control,
- for multi-media, real-time applications executing on parallel and distributed target computing platforms.

More specifically, DIMAS will address the issues of system response to dynamic external environments in the domain of interactive sound and video generation, processing, and display. Traditional computer based multimedia generation systems have the following shortcomings that limit flexibilty and creativity:

- Typically the sound and graphics generation algorithms (the patch) and score are fixed before the playing of the piece. Little flexibility in terms of modifying patches or the play sequence of a piece are provided. This preprogrammability dampens the ability of the user to creatively change in any significant way the format or texture of a piece once it has been started.
- Computer based sound generation systems only allow the interaction of a single user by way of a console or keyboard. This limitation is not necessary, since multiple users may wish to interact with the piece as it plays.

- Most production computer based multimedia generation systems are embedded in hardware with limited patch manipulation capabilities. As a result, the typical studio will have several different types of computer generated multimedia equipment in order to produce the different types of sounds necessary to produce a piece.

- Some computer based music generation systems, such as Csound and Cmusic, are software based. However, the computational complexity of many of the sounds these systems generate prohibits real-time usage of these systems. Exploitation of multiprocessor technologies may remove this limitation.

DIMAS addresses the shortfalls for computer based multimedia generation. DIMAS offers dynamic interactions with external "multimedia agents" during the execution of a piece. As an example, consider the following two agents and their on-line interactions with DIMAS.

Performance agents. A performance agent issues a sequence of musical notes or images to be generated. Typically, this type of agent is a human playing a MIDI-capable instrument or a computer based program that generates MIDI sequences. Performance agents can also produce a series of video or graphics images to be integrated with and synchronized into a musical piece. Two types of interactions exist between DIMAS and a performance agent. First, the performance agent can initiate changes in tempo, exchange or modify patches, volume, and pitch. Second, DIMAS provides system status indicators to human proformance agents via a console and to computer based performance agents via a message facility. The message facility permits multimedia agents to interact with one another. For example, a human performance agent can ask for a certain section of a piece to be repeated. DIMAS will send a corresponding message to each of the other human and computer based performance agents with the information indicating what section and when the section will be repeated.

Production agents. A production agent will manage the overall production of a piece by giving instructions to both DIMAS and the performing agents. The types of instructions that performance agents issue include but are not limited to:

- Patch modification of a certain sound.

- Patch changes.

- Repeat or change to a new section of a piece.

- Change tempo or pitch.

- Adding and modifying effects such as chorusing and reverberation.

- Syncronizing video and graphic images.

- Control of volume, panning, and mixing of tracks generated by performance agents.

DIMAS will accept messages that affect it directly (such as patch change or modification) and also forward production agent messages to performance agents (change tempo or pitch, repeat or jump to a new section). In summary, DIMAS provides an environment that provides flexibility in the way in which a multimedia piece is played, by allowing changes in the execution environment while the piece is playing.

2 DIMAS Components

DIMAS Objects. DIMAS structures interactive multimedia applications as sets of interacting objects, which implement the performance and production agents mentioned above. This is a natural representation for many multimedia applications since they are easily regarded as collections of "black boxes" performing computations and communicating with one another. For example, a typical real-time sound synthesis application can be broken down into components such as oscillators, mixers, and filters, which communicate with one another via sample passing. In addition, many of today's multimedia applications represent the unification in one computer program of jobs that were formerly performed by separate hardware devices connected by wires. For example, the real-time synthesis application mentioned above is the natural descendant of early synthesizers which contained many discrete hardware oscillators, filters, and so on, connected by patch cables.

The "black boxes" mentioned above translate naturally into program objects. The connections between these "black boxes" translate into interactions between program objects. Furthermore, the object-based approach affords a straightforward approach to parallel programming, since program objects are a natural unit of computation to distribute among multiple processors. Finally, program objects and their interactions can be associated with timing attributes in a straightforward way[11, 7]. As a result, any DIMAS application is constructed from a set of audio and graphics generation objects.

Links and attributes. *Links* embody the interaction semantics between objects. Namely, program object instances are connected by link instances, and links contain details descriptions of user-defined invocation semantics. For example, simple link semantics are synchronous message passing, asynchronous message passing, and conventional object invocation. More complicated examples include remote object invocation, video frame exchange, and emulation of the MIDI protocol.

Invocations among DIMAS objects implementing multi-media applications are often subject to timing constraints. Since such constraints must be explicitly specified and enforced, DIMAS program objects are not just characterized by their classes, but also support extensive compile-, load-, and runtime *attributes*, including invocation time attributes. These attributes can be interpreted in a variety of ways by the invoked object; typically, they are used for specification of timing constraints on the execution of objects' code and for determination of assignments of objects to processors.

From the point of view of an object which uses links to interact with other objects, the links are themselves objects which export methods that implement the interaction. For example, 'SamplesLink' is an asynchronous message passing link used for exchanging packets of samples in digital audio applications. It exports send() and receive() methods.

As might be suggested by the above, links are often implemented as objects. Such is the case with 'SamplesLink'. However, links do not have to be implemented as objects. For example, a link whose job is to simply pass conventional object method invocations on to another object can be implemented by rerouting the invocation at compile time to the appropriate destination.

A primary advantage of the link concept is that it allows the programmer to avoid explicitly naming other object instances in the description of an object class which is intended for reusability. An example is a 'Mixer' object, which uses a variable number of 'SamplesLinks' to perform its input, and one 'SamplesLink' to perform its output. The 'Mixer' object thus implements a computational "black box" which is quite analogous to a real hardware mixer. Its job is to do mixing; it does not have any knowledge of the objects which produce its inputs or consume its outputs.

Constructing DIMAS applications. Constructing a DIMAS application (a multimedia artifact) can be thought of as proceeding in several phases. In the first phase, the programmer writes class descriptions for any required new program object classes, and in some cases for new link types. In the second phase, the programmer uses language-level tools to create program object instances, connect these instances using instances of various types of links, and assign and analyze timing attributes and parallel execution attributes. In the third phase, the programmer adds user-interactive object instances, thereby essentially designing an application-specific user interface. Finally, in the fourth phase, the programmer uses graphical or MIDI-capable user interfaces to interact with the running DIMAS program.

DIMAS application development is simpler than the development of arbitrary parallel object-based programs because built into the DIMAS system are a collection of audio and graphics generation objects. A piece is constructed by assembling a group of audio instruments and video generators. These instruments will produce sound/images based on event inputs from the performance agents external to the DIMAS system. Both performance and production agents can modify the parameters of these objects be invoking their 'change' methods.

The DIMAS runtime system. The two primary parts of the DIMAS system, therefore, are (1) the environment for defining and instantiating new classes of DIMAS objects outlined above (using an object-oriented language and graphical user interface), and (2) runtime system support for the efficient and parallel execution of DIMAS objects. Runtime support includes facilities for the efficient on-line monitoring of DIMAS applications[9], for representation of DIMAS objects, and for real-time dynamic scheduling – The Dynamic Parallel Adaptive Real-Time Scheduler (DPARTS). The DPARTS scheduler is described next. More general operating system support for real-time objects is described elsewhere[10, 7].

DPARTS will schedule and invoke groups of generation objects based on event inputs from performance agents. DPARTS is adaptive because it will modify the schedule based on the available time and resources that the generation objects have to do their jobs. Because of this feature DPARTS may schedule only certain features of a generative object if the time and/or resources are not available for a complete generation. Part of DPARTS task is to provide information to production agents about resources available and when DPARTS uses the reduced functionality of a generative object based on limited resources. Production agents can send messages to DPARTS indicating on what priority can generative objects can be modified or even eliminated based on resource availablity.

In order to achieve real-time response, the DPARTS scheduler will be internally concurrent so it can be easily scaled to different size parallel machines and to varying application demands. This implies that scheduling will be performed by multiple concurrent and cooperating tasks. Furthermore, since large-scale machines typically exhibit non-uniform memory access (NUMA) characteristics (or do not offer shared memory at all, as with distributed memory machines), the DPARTS scheduler will be designed to access scheduling information locally whenever possible. In other words, its scheduling information will be distributed across multiple memory units. Such distribution of scheduling information enables access locality and concurrency during scheduling. However, additional scheduling overhead may result from accesses to remote scheduling information required in several situations, including when a processor's scheduler attempts to schedule a task on a remote processor. Such overhead can be reduced by the scheduler's use of incomplete information about other processors' schedules[14].

Some details about the DPARTS scheduler further demonstrate its novelty. As stated above, each external event may trigger a sequence of processes that jointly handle the event. These processes must execute in an application-specific order. DPARTS must have information about (1) such orderings, (2) the computation time of the involved processes, and (3) about possible alternate or optional processes. We will develop a directed graph representation to contain such information, where the computation times and any other information DPARTS needs to know about processes is stored in the graph's nodes, and the directed edges represent the processes' order of computation. Processes must be scheduled concurrently and to meet real-time constraints, such that the graph's topological order is maintained. This implies that multiple parts of the schedule graph may be active at any point during the applications execution.

The specific research questions to be addressed in this work focus on the development of fast heuristic algorithms, on experimentation with those algorithms, and on the use of such algorithms within the music application described above. We will not develop optimal algorithms since all of the scheduling and assignment problems we are addressing have been shown NP-hard[12]. Some specific questions we intend to address include:

1. Deadline semantics – in our sample music application, precise deadlines are usually not necessary in order to produce acceptable results. As such we will evaluate what type

of sematics are appropriate for addressing the application we are scheduling.

2. Should DPARTS have the option of rescheduling existing tasks if such rescheduling can result in the successful scheduling of otherwise unschedulable process sequences?

3 Status, Conclusions, and Related Research

A current prototype DIMAS program runs on SparcStations, on the BBN Butterfly GP1000 multiprocessor, and is being ported to a 64-node KSR2 parallel supercomputer for attainment of real-time response for complex on-line sound generation. In addition, multiprocessor SGI machines will provide the sounds and graphics capabilities required by realistic DIMAS artifacts. DIMAS is being constructed using two existing platforms for the construction of real-time applications: real-time threads[13] and the Chaos-arc real-time operating system[7].

The next steps in our work concern construction of a more complex sound synthesis application, followed by its use for a first prototype of the DPARTS runtime scheduler. Experimentation with the resulting multimedia program should establish whether real-time response to human-generated MIDI events can be attained on current parallel machines. Once we have established that fact, we will continue our work by construction of a more general DIMAS system, its user interfaces, and its applications.

Our work differs from other research in multimedia systems in that most current multimedia operating system efforts are focusing on the support of frame-based interactive video, in file systems, with runtime scheduling or resource allocation[3, 2, 1], and in communication protocols that attempt to reserve end to end channel bandwidth. In contrast to such work, the focus of DIMAS is to allow dynamic configuration and control of multimedia streams in real-time. Both internal heuristics and external interfaces to the system allow for modifications of generation algorithms based on system resources.

Acknowledgements. David Rosenbaum contributed extensively both to earlier drafts of this paper and to the DIMAS concepts and our DIMAS execution environment. This work is supported in part by the Office of Naval Research under grant #N00014-93-1-0783.

References

[1] D.P. Anderson and R.G. Herrtwich. Resource management for digital audio and video. In *Seventh IEEE Workshop on Real-Time Operating Systems and Software*, pages 99–103. IEEE, May 1990.

[2] D.P. Anderson and R. Kuivila. A system for computer music performance. *ACM Transactions on Computer Systems*, 8(1):56–82, Feb 1990.

[3] D.P. Anderson, S. Tzou, R. Wahbe, R. Govindan, , and M. Andrews. Support for continous media in the dash system. In *Proceedings of the 10th International Conference on Distributed Computing Systems, Paris France.* IEEE,ACM, 1990.

[4] T. Bihari and K. Schwan. Dynamic adaptation of real-time software. *ACM Transactions on Computer Systems*, 9(2):143–174, May 1991. Older version available from the Department of Computer and Information Science, The Ohio State University, OSU-CISRC-5/88-TR, newer version available from College of Computing, Georgia Institute of Technology, Atlanta GA, GTRC-TR-90/67.

[5] S. Davari, T.F. Leibfried, S. Natarajan, D.Pruett, L. Sha, and W. Zhao. Real-time issues in the design of data management for the space station freedom. In *Workshop on Real-Time Applications, New York*, pages 161–165. IEEE, May 1993.

[6] Ed Ferguson. Resource scheduling for adaptive systems. In *Workshop on Real-Time Applications, New York*, pages 102–103. IEEE, May 1993.

[7] Ahmed Gheith and Karsten Schwan. Chaos-arc – kernel support for multi-weight objects, invocations, and atomicity in real-time applications. *ACM Transactions on Computer Systems*, 11(1):33–72, April 1993.

[8] Jim McDonald and Karsten Schwan. Ada dynamic load control mechanisms for distributed embedded battle management systems. In *First Workshop on Real-time Applications, New York*, pages 156–160. IEEE, May 1993.

[9] David M. Ogle, Karsten Schwan, and Richard Snodgrass. The dynamic monitoring of real-time distributed and parallel systems. Technical report, College of Computing, Georgia Institute of Technology, ICS-GIT-90/23, Atlanta, GA 30332, May 1990. To appear in IEEE TPDS August 1993.

[10] Karsten Schwan, Prabha Gopinath, and Win Bo. Chaos – kernel support for objects in the real-time domain. *IEEE Transactions on Computers*, C-36(8):904–916, July 1987.

[11] Karsten Schwan, Rajiv Ramnath, Sridhar Vasudevan, and Dave Ogle. A system for parallel programming. In *9th International Conference on Software Engineering, Monterey, CA*, pages 270–282. IEEE, ACM, March 1987. Awarded best paper.

[12] Karsten Schwan and Hongyi Zhou. Dynamic scheduling of hard real-time tasks and real-time threads. *IEEE Transactions on Software Engineering*, 18(8):736–748, Aug. 1992.

[13] Karsten Schwan, Hongyi Zhou, and Ahmed Gheith. Multiprocessor real-time threads. *Operating Systems Review*, 25(4):35–46, Oct. 1991. Also appears in the Jan. 1992 issue of Operating Systems Review.

[14] Hongyi Zhou, Karsten Schwan, and Ian Akyildiz. Performance effects of information sharing in a distributed multiprocessor real-time scheduler. Technical report, College

of Computing, Georgia Tech, GIT-CC-91/40, Sept. 1991. Abbreviated version in 1992 IEEE Real-Time Systems Symposium, Phoenix.

PARSYNTH: A Case Study on Implementing a Real-Time Digital Audio Synthesizer

Byron Jeff
Department of Computer and Information Science
Clark Atlanta University
Atlanta, GA 30314
Email: byron@cc.gatech.edu
Phone: 404-880-6934
Fax: 404-880-6963

Karsten Schwan
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332
Email: schwan@cc.gatech.edu
Phone: 404-894-2589

Keywords: Digital Audio Synthesis, Multimedia, Dynamic Scheduling

1 Motivation

Real-time systems interact with their external execution environments, and complex, large-scale systems often cannot anticipate all such interactions. As a result, real-time systems are increasingly being constructed with mechanisms and policies that permit them to adjust to new environmental conditions, based on higher-level knowledge about system functionality and requirements[6, 1, 3, 2].

In addition the area of real-time scheduling has received increased attention during the last few years, in part due to the creation of many recent multiprocessor and distributed applications demanding real-time performance. Such applications include robotics , multi-media, and music . These applications are characterized by their need for real time response, heavy computational load, and unpredictable events sequences that must be processed in a timely manner.

The specific application domain addressed by this research, Dynamic Interactive Multimedia ArtifactS (DIMAS), concerns multi-media systems. Specifically, DIMAS will provide a framework for the real-time synthesis of voice, music, graphics, and video where partially ordered sequences of computations and communications must be processed under given real-time constraints. This framework will consist of:

- A library of multimedia generation and manipulation objects and the linkages between these objects.
- Language support for specifying interconnections between existing library objects and the ability to create user defined application specific objects and links.
- Run time support to execute a DIMAS constructed application in real-time on shared memory multiprocessor computers.

DIMAS will utilize the Dynamic PArallel Real-Time Scheduler (DPARTS) to perform (on-line) scheduling of action sequences triggered by real-time events, where multiple action sequences and different independent segments of a single action may execute in parallel.

The PARSYNTH real-time parallel digital audio synthesis system is an application using the DIMAS run-time libraries. This concrete example of a real-time application, provides a testbed for testing DIMAS/DPARTS mechanisms and policies.

2 DIMAS SYSTEM OVERVIEW

The goals of the DIMAS system are:

- To provide mechanisms, policies, and human interfaces for dynamic system configuration and control,

- To provide a framework for the execution of multi-media, real-time applications on parallel and distributed target computing platforms.

More specifically, DIMAS will address the issues of system response to dynamic external environments in the domain of interactive sound, video, and graphics generation, processing, and display.

Traditional computer based multimedia generation systems have shortcomings that limit flexibilty and creativity [5]. These fall into two broad categories:

- Hardware based generation systems have fixed generation algorithm (patch) capability. There are limits to the number of available patches, the number of events that the hardware device can simulteaneosly process, and the types of manipulations that can be performed on individual patches during the execution of a piece. Hardware based system however do provide real-time response to events within their limited scope.
- Software based generation systems do provide more flexibility in terms of the number of sounds per patch and patch manipulation during execution. However the computational complexity of such systems fail to deliver real-time performance levels for all but the most simple compositions.

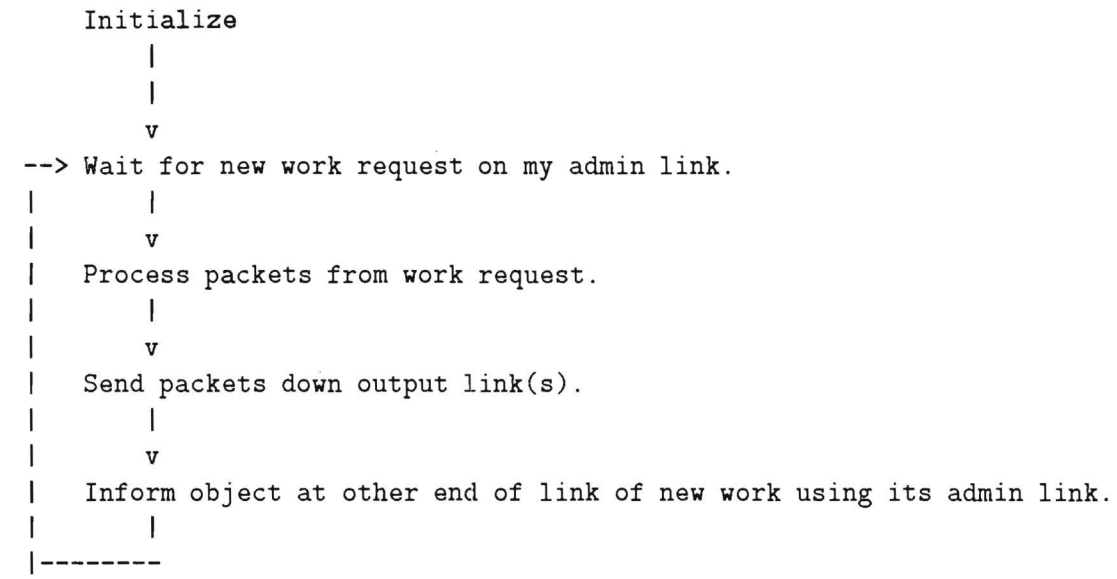
2.1 DIMAS Components

2.1.1 DIMAS Objects

DIMAS structures interactive multimedia applications as sets of interacting objects, which implement the performance and production agents mentioned above. This is a natural representation for many multimedia applications since they are easily regarded as collections of "black boxes" performing computations and communicating with one another. For example, a typical real-time sound synthesis application can be broken down into components such as oscillators, mixers, and filters, which communicate with one another via sample passing. In addition, many of today's multimedia applications represent the unification in one computer program of jobs that were formerly performed by separate hardware devices connected by wires. For example, the real-time synthesis application mentioned above is the natural descendant of early synthesizers which contained many discrete hardware oscillators, filters, and so on, connected by patch cables.

The "black boxes" mentioned above translate naturally into program objects. The connections between these "black boxes" translate into interactions between program objects. Furthermore, the object-based approach affords a straightforward approach to parallel programming, since program objects are a natural unit of computation to distribute among multiple processors. Finally, program objects and their interactions can be associated with timing attributes in a straightforward way[7, 4]. As a result, DIMAS applications are constructed from a set of audio and graphics generation objects.

To maintain flexibility for execution each object is designed to process a small amount of work for each work request. The basic model for a object is as follows:



A DIMAS object will accept work from any entity that is able to send administrative packets to the admin link for the object. This provides a simple linkage mechanism for the DPARTS scheduler to control DIMAS objects. The scheduler will be able to send work requests to each of the objects specifying the amount of work each object is to perform. CPU utilization for each objects can be varied by specifying differents amounts of work for each iteration through the execution loop. In addition the scheduler can receive information from objects via a simularly constructed administrative link. The object execution loop model is designed to facilitate the interfacing of DIMAS objects and the DPARTS scheduler.

2.1.2 DIMAS Links

Links embody the interaction semantics between objects. Namely, program object instances are connected by link instances, and links contain detailed descriptions of user-defined invocation semantics. For example, simple link semantics are synchronous message passing, asynchronous message passing, and conventional object invocation. More complicated examples include remote object invocation, video frame exchange, and emulation of the MIDI protocol.

From the point of view of an object which uses links to interact with other objects, the links are themselves objects which export methods that implement the interaction. For example, 'SamplesLink' is an asynchronous message passing link used for exchanging packets of samples in digital audio applications. It exports *send()* and *receive()* methods. Simple links however can be implmented without defining specific objects but by simply rerouting the invocation/message from one DIMAS object to another.

A primary advantage of the link concept is that it allows the programmer to avoid explicitly naming other object instances in the description of an object class which is intended for reusability. An example

is a 'Mixer' object, which uses a variable number of 'SamplesLinks' to perform its input, and one 'SamplesLink' to perform its output. The 'Mixer' object thus implements a computational "black box" which is quite analogous to a real hardware mixer. Its job is to do mixing; it does not have any knowledge of the objects which produce its inputs or consume its outputs. In addition the 'Mixer object' can dynamically be instructed to accept or drop additional links at runtime.

2.2 Constructing DIMAS applications

Constructing a DIMAS application can be thought of as proceeding in several phases:

Phase 1 - the programmer writes class descriptions for any required new program object classes, and in some cases for new link types.

Phase 2 - the programmer uses language-level tools to create program object instances, connect these instances using instances of various types of links, and assign and analyze timing attributes and parallel execution attributes.

Phase 3 the programmer adds user-interactive object instances, thereby essentially designing an application-specific user interface.

Phase 4 the programmer uses graphical or MIDI-capable user interface DIMAS objects to interact with the running DIMAS program.

DIMAS application development is simpler than the development of arbitrary parallel object-based programs because built into the DIMAS system are a collection of audio and graphics generation objects along with graphical and device interface objects. A composition is constructed by assembling a group of audio-video generator objects. These objects will produce sound/images based on event inputs from input devices external to the DIMAS application which interface to the application via the DIMAS device interface objects. External users can interact with the DIMAS application via DIMAS user interface objects.

Another aspect of the DIMAS system paradigm is that once system performance are consistently maintaining real-time levels that the actual artifact generated by the system need not be maintained by the system after execution. Since the specifications of a composition (including objects, patches, events, scores, visual objects, links) are often several orders of magnitude smaller in size than the artifact produced by the execution of the system, a significant reduction in the storage and transmission requirements of a composition can be achieved.

3 A DIMAS APPLICATION: PARSYNTH

A parallel digital audio synthesizer (PARSYNTH) serves as an application implementation using the DIMAS run-time libraries. PARSYNTH is an example of the type of application produced at the end of the DIMAS application specification compilation phase describe above. PARSYNTH has been implemented with a message passing link mechanism that represents the links in the DIMAS model. PARSYNTH

also implements the DIMAS object model that simplifies the linkage of the PARSYNTH application to the DPARTS scheduler. The PARSYNTH implementation serves as a testbed for demonstrating concepts and models embodied within the DIMAS system.

PARSYNTH requires several audio representations to do its job. PARSYNTH produces CD quality audio at a sample rate of 44.1 Khz. The digital audio entities based on this representations are:

- sample - a single unit of digital audio. As stated above PARSYNTH must product 44100 samples for each second of sound generated.
- packet - The smallest unit of digital audio that PARSYNTH uses. A packet currently consist of 21 samples. PARSYNTH produces 2100 packets per second. All of PARSYNTH's timings are based on packet numbers.

3.1 PARSYNTH IMPLEMENTATION

3.1.1 COMPUTATIONAL IMPLMENTATION

PARSYNTH is implemented with the DIMAS object run-time libraries.

The C-threads parallel threads library provides a lightweight threads environment for implementing objects. The DIMAS run-time object library is implemented in C-threads. Each object is assigned to a single thread with multiple threads assigned to each available processor in the multiprocessor machine.

3.1.2 OBJECT TYPES

Currently 5 types of objects are implemented:

Generators: This type of thread generates a digital audio stream. Like all objects generators sit idle until a NOTE ON work request come through its admin link. At this point a generator will produce a small number of packets for the note, generate a NEW WORK event for the downstream object, then wait on its admin link. Generally a CONTINUE message from the downstream object will come along and the generator will produce another set of packets. This process continues until a NOTE OFF work reqest comes though. The NOTE OFF request is passed downstream and the generator discontinues the generation of packets and goes back into its idle state, waiting for another work request.

Syncronizer: This type of object is required when multiple audio links are directed into a single object. Since generators function asyncronously and there is some latency involved between the receipt of a work request and the actual generation of packets, often the packet numbers between different streams vary. This object's job is to receive multiple links of input, collate the links into syncronization by matching the packet numbers of the packets on each link , and pass the syncronized packets of each link into the output links. Not only can a syncronizer object have multiple input and output links, but these links are dynamically allocated and destroyed by NOTE ON/OFF messages sent to the admin

link. These objects must generate CONTINUE messages for each of the upstream objects and a NEW WORK message for the downstream objects. Also it receives and processes NEW WORK messages from the upstream objects.

Combiner: This type of object combines multiple links into a single link. By convention the incoming links must be syncronized by a syncronizer object. Typically a Combiner object will have a single output link that the combined results of the input. As with other objects a combiner object creates and destroys input links on NOTE ON/NOTE OFF messages, accepts new work from NEW WORK messages coming from upstream, sends NEW WORK messages downstream with whatever new work it generates, and accepts CONTINUE messages from the downstream object.

The first implemented combiner object is an digital audio mixer.

Output: This object type has the specific purpose of directing the final digital audio link produced from PARSYNTH to the outside world. Typically there will only be one output object in a PARSYNTH execution. This object has the potential of having multiple input links, but obviously output links are out. Output objects respond to the typical NOTE ON / NOTE OFF / NEW WORK / CONTINUE admin messages as other types of objects. However as the single point of having a complete audio stream, output objects monitors the packet numbers of the output stream and sends reports to the dispatch object (described below) creating a closed system that self governs its production rate.

Due to the large size of the output streams generated, the implemented output object actually sends the stream to a Unix socket. An external applications can then direct the stream to the digital audio output device or to a file.

An output device that discards the output is currectly under development. This output object is useful for sample runs where the output isn't relavent.

Dispatch: This type object is unique in that it doesn't directly process any of the digial audio streams in parsynth. Dispatch object performs the following:

- - Monitor for new incoming events
- - Introduce new events into the PARSYNTH system
- - Monitor the packet numbers output from the output object

Dispatch interacts with the rest of the PARSYNTH objects solely through the admin links. It injects NOTE ON and NOTE OFF messages into appropriate generators to create/destroy links for new events.

3.1.3 COMMUNICATIONS IMPLEMENTATION

Objects in PARSYNTH transmit data between one another using links. PARSYNTH uses shared memory, which is implemented in the C-Threads Library, to implement both packets and links. Inter-object communication is accomplished through a separate link called the admin link. Each object

receives work requests via its admin link. Objects communicate with other objects by sending messages to the destination object's admin link. The types of admin messages sent are as follows:

- NOTE ON - This message signals the start of a new note. The event causes the dynamic reconfiguration of PARSYNTH by adding a new set of links through which the system process the event.
- NOTE OFF - This message signals the end of a note. The event causes the reconfiguration of PARSYNTH by destroying the set of links that the NOTE OFF was being processed.
- NEW WORK - This message accompanies any new work processed by an object. It is sent "downstream" to the next object to signal it to begin work.
- CONTINUE - This is an acknowledgement message that acknowledges a NEW WORK admin message. This message is sent "upstream" after the new work has been processed.
- INFO - This message is used to alter the internal state of a object. Its primary purpose is to change the virtual chain of admin links.
- DIE - This message is used to signal an object to cease operations.

4 DIMAS Run-time Scheduler: DPARTS

In order for DIMAS applications to achieve real-time performance under varying external event input loads, a dynamic scheduling system must be utilized at run time. The DIMAS framework along with the underlaying DPARTS parallel real-time scheduler will utilize the enhanced computational power of parallel distributed computing platforms to give real-time performance for software based multimedia applications.

In addition DPARTS will utilize application specific information in order to make scheduling decisions. For example in digital audio applications perceived lateness is tied to the initial attack rate of a note. For example sounds with fast attack rates, such as percussive and piano sounds, will be perceived as being late sooner than a softer attack, such as strings. The DIMAS application designer can embody such attributes in the object specification and the DIMAS objects will share this scheduling information with the DPARTS scheduler. So in the instance that an event with a hard attack rate and one with a soft attack rate are presented to the DPARTS scheduler at the same time, the scheduler will schedule the event connected to the hard attack rate before the event with the soft attack. Even in the case where the soft attack event misses its dealine, the perception of lateness by the listener will be lessened.

DPARTS will schedule and invoke groups of DIMAS objects based on event inputs from DIMAS objects that interface to devices external to the DIMAS application. DPARTS must be dynamic because it will have to modify the schedule based on the available time and resources that the DIMAS objects have to do their jobs. Because of this feature DPARTS may schedule only certain features of a DIMAS object if the time and/or resources are not available for a completion of that object's task. Part of DPARTS

task is to provide information to DIMAS objects about resources available and instruct DIMAS objects to use reduced functionality because of limited resources.

In order to achieve real-time response, the DPARTS scheduler will be internally concurrent so it can be easily scaled to different size parallel machines and to varying application demands. This implies that scheduling will be performed by multiple concurrent and cooperating objects.

As stated above, each external event may trigger a sequence of objects that jointly process the event. These objects must execute in an application-specific order. DPARTS must have information about (1) such orderings, (2) the computation time of the involved objects, and (3) about possible alternate or optional object invocations. We will develop a directed graph representation to contain such information, where the computation times and any other information DPARTS needs to know about objects is stored in the graph's nodes, and the directed edges represent the processes' order of computation. Objects must be scheduled concurrently and to meet real-time constraints, such that the graph's topological order is maintained. This implies that multiple parts of the schedule graph may be active at any point during the applications execution.

5 Initial Results

The current PARSYNTH application runs on three platforms:

- Pentium 90 Mhz running Linux 1.2.13
- Sun Sparc running SunOS 4.1.3
- SGI Challenge with 4 MIPS 4400 150 Mhz CPUs running IRIX 5.3

Currently PARSYNTH's dispatch object simulates the generation of events from MIDI devices by reading from MIDI files that have been translated into packet event format. In order to guage the relative performance times of the PARSYNTH application, events from these files are passed into PARSYNTH system as fast as the system can process the events even if the input rate exceeds actual real-time.

All testing was performed on the SGI platform using a 4 instrument MIDI file that generates 60 seconds of digital audio. The PARSYNTH application consists of 8 objects: 4 sine wave generators, a syncronizer, a mixer, an output object, and the dispatch object. For testing purposes the final digital audio stream are not written into a file.

The initial result using a single processor multi-thread execution run completed in 31.5 seconds. However the initial run of the 4 processor multi-thread execution run took 234.31 seconds to complete.

An examination of the trace file generated from Cthread's monitoring facilities showed that for the multiprocessor run that the threads processors were idle much of the time. This idleness was due to the light load the input event file provided to the system. The lack of events to process resulted in most of the 8 objects blocking on links waiting for work.

As an initial test of this hypothsis the packet size for PARSYNTH was increased by a factor of 10 (from 21 samples per packet to 210 samples per packet). The change in packet size results in a factor of 10 change to the computation communication cycle that is designed into each PARSYNTH object.

The results were improved. The single processor multi-thread execution run completed in 17.5 seconds which is a 45 percent reduction in the completion time for the the run. The 4 processor multi-thread execution run dropped from 234.31 seconds to 41.47 seconds which is an 83.4 percent reduction in the completion time.

These results point towards three trends:

- The overall performance of the system can be controlled by varying the grainularity of the packet size. Larger packet sizes creates more computation and less communication while smaller packet sizes results in more communications.
- The multiprocess application needs heavier workloads to be effective.
- Some fine tuning is required to reduce the computation time of the multiprocessor application to match and exceed the computation time for the single processor version.

6 Next Steps

Further areas of interest are:

- Implement the DPARTS scheduler and use it to schedule the objects for DIMAS applications such as PARSYNTH.
- Fine tune the multiprocessor execution environment for parsynth.
- Test PARSYNTH under varying loads in terms of packet rate for each instrument and the number of intruments to observe PARSYNTH's performance.
- Implement a dispatch object that actually monitors a live MIDI keyboard for input events.

7 Acknowledgements

Vernard Martin assisted in editing and with the monitoring of PARSYNTH. This work is supported in part by the Office of Naval Research under grant N00014-93-1-0783 and in part by ARO grant # DAAL03-92-G-0377.

References

[1] T. Bihari and K. Schwan. Dynamic adaptation of real-time software. *ACM Transactions on Computer Systems*, 9(2):143–174, May 1991. Older version available from the Department of Computer and Information Science, The Ohio State University, OSU-CISRC-5/88-TR, newer version available from College of Computing, Georgia Institute of Technology, Atlanta GA, GTRC-TR-90/67.

[2] S. Davari, T.F. Leibfried, S. Natarajan, D.Pruett, L. Sha, and W. Zhao. Real-time issues in the design of data management for the space station freedom. In *Workshop on Real-Time Applications, New York*, pages 161–165. IEEE, May 1993.

[3] Ed Ferguson. Resource scheduling for adaptive systems. In *Workshop on Real-Time Applications, New York*, pages 102–103. IEEE, May 1993.

[4] Ahmed Gheith and Karsten Schwan. Chaos-arc – kernel support for multi-weight objects, invocations, and atomicity in real-time applications. *ACM Transactions on Computer Systems*, 11(1):33–72, April 1993.

[5] B. Jeff and K. Schwan. Dimas-constructing dynamic interactive multimedia artifacts. In *Workshop Proceedings: THE ROLE OF REAL-TIME IN MULTIMEDIA/INTERACTIVE SYSTEMS.* IEEE, 1993.

[6] Jim McDonald and Karsten Schwan. Ada dynamic load control mechanisms for distributed embedded battle management systems. In *First Workshop on Real-time Applications, New York*, pages 156–160. IEEE, May 1993.

[7] Karsten Schwan, Rajiv Ramnath, Sridhar Vasudevan, and Dave Ogle. A system for parallel programming. In *9th International Conference on Software Engineering, Monterey, CA*, pages 270–282. IEEE, ACM, March 1987. Awarded best paper.

PRINCIPAL INVESTIGATOR K SCHWAN CENTER NO. 246R79170A0 ACCOUNT NO. C-36-X17

STATUS AT END OF JULY 1996 DEPARTMENT COLL COMP

SPONSOR NAVY

AWARD NUMBER N00014-93-1-0783 RF CENTER NO. 00642901000 RESTRICTED FUND RF-48202

EFFECTIVE DATE 07-01-93 BILLING GROUP GTRC EXPIRATION DATE 06-15-96

	MONTH	FISCAL YEAR	TOTAL CONTRACT
PERSONAL SERVICES			
BUDGET			36,829.00
EXPENDED	.00	.00	38,007.99
ENCUMBERED	.00	.00	.00
FREE BALANCE			-1,178.99
MATERIALS AND SUPPLIES			
BUDGET			3,579.00
EXPENDED	.00	.00	3,470.01
ENCUMBERED	3,184.00	3,184.00	3,184.00
FREE BALANCE			-3,075.01
TRAVEL			
BUDGET			3,878.00
EXPENDED	.00	.00	686.86
ENCUMBERED	.00	.00	.00
FREE BALANCE			3,191.14
CAPITAL OUTLAY			
BUDGET			3,500.00
EXPENDED	.00	.00	3,881.00
ENCUMBERED	.00	.00	.00
FREE BALANCE			-381.00
TOTAL DIRECT CHARGES			
BUDGET			47,786.00
EXPENDED	.00	.00	46,045.86
ENCUMBERED	3,184.00	3,184.00	3,184.00
FREE BALANCE			-1,443.86

PRINCIPAL INVESTIGATOR K SCHWAN CENTER NO. 246R7917OAO ACCOUNT NO. C-36-X17

STATUS AT END OF JULY 1996 DEPARTMENT COLL COMP

SPONSOR NAVY

AWARD NUMBER N000l4-93-1-0783 RF CENTER NO. 00642901000 RESTRICTED FUND RF-48202

EFFECTIVE DATE 07-01-93 BILLING GROUP GTRC EXPIRATION DATE 06-15-96

	MONTH	FISCAL YEAR	TOTAL CONTRACT	
INDIRECT CHARGES				
BUDGET			20,580.00	
EXPENDED	.00	.00	16,997.88	RATE OF 49.5 BASE OF 4
ENCUMBERED	1,576.08	1,576.08	1,576.08	
FREE BALANCE			2,006.04	
TOTAL				
BUDGET			68,366.00	
EXPENDED	.00	.00	63,043.74	THIS ACCOUNT IS 99.2 EXPENDED
ENCUMBERED	4,760.08	4,760.08	4,760.08	
FREE BALANCE			562.18	